Engineering with Computers (2005) 21: 76–90
DOI 10.1007/s00366-005-0002-x

SPECIAL ISSUE

David R. White · Sunil Saigal · Steven J. Owen

Meshing complexity: predicting meshing difficulty for single part CAD models

Received: 14 September 2003 / Accepted: 13 October 2004 / Published online: 8 October 2005

Abstract This paper proposes a method for predicting the complexity of meshing computer aided design (CAD) geometries with unstructured, hexahedral, finite elements. Meshing complexity refers to the relative level of effort required to generate a valid finite element mesh on a given CAD geometry. A function is proposed to approximate the meshing complexity for single part CAD models. The function is dependent on a user defined element size as well as on data extracted from the geometry and topology of the CAD part. Several geometry and topology measures are proposed, which both characterize the shape of the CAD part and detect configurations that complicate mesh generation. Based on a test suite of CAD models, the function is demonstrated to be accurate within a certain range of error. The solution proposed here is intended to provide managers and users of meshing software a method of predicting the difficulty in meshing a CAD model. This will enable them to make decisions about model simplification and analysis approaches prior to mesh generation.

Keywords Time to mesh · Mesh complexity · Blend detection · Geometry clean-up

D. R. White (✉) · S. J. Owen · S. Saigal
Department of Civil and Environmental Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213, USA

D. R. White (✉) · S. J. Owen
Sandia National Laboratories, PO Box 5800, MS 0847,
Albuquerque, NM, 87185-0847, USA
E-mail: drwhite@sandia.gov
Fax: +505-8449297

S. Saigal
Department of Civil and Environmental Engineering,
University of South Florida,
Tampa, FL 33620-5350, USA

1 Introduction

Mesh generation is often the most time consuming part of finite element-based computer simulations. Research in mesh generation has attempted to remedy this issue with varying degrees of success. A main difficulty in solving the meshing issue is the extent of the problem space. Mesh generation software is expected to automatically create meshes for any shape created by design engineers or other upstream sources. When the set of shapes given to the mesh generator is restricted, automatic mesh generation is indeed possible. Unfortunately, the problem space cannot usually be restricted.

Currently, shapes or computer aided design (CAD) models are given to analysts to perform the computational simulation. While meshing a model, the analyst frequently finds problems that must be fixed by the design engineer who created the model. Once those problems are fixed, new problems may be found, and an iterative process ensues until a mesh is finally generated. At times it may even be necessary to discard the design model and build a new model more amenable to mesh generation. After the model is *cleaned*, or its problems removed, it can usually be meshed automatically with tetrahedral or mixed elements. If the analysis dictates all hexahedral elements, additional geometry manipulations may be required before the mesh can be generated. This is due to the increased complexity of generating a boundary-conforming all-hexahedral mesh for arbitrary geometry. Current state-of-the-art for reliable hexahedral mesh generation dictates that volumes be decomposed such that sweeping or two and one-half dimensional (2.5 D) algorithms be employed. For tetrahedral mesh generation, model *cleanliness* typically dictates success in meshing, while for hexahedral meshing both *cleanliness* and *sweep-ability* contribute to its success. Cleanliness of the model and the ease with which a given meshing algorithm can mesh it contribute to the goodness or *complexity* of the model. *Meshing complexity* for a given model serves as the measure of difficulty required for an analyst to generate a valid mesh for that model.

This paper describes a method to quantify the meshing complexity of CAD models for the purpose of unstructured hexahedral mesh generation. Although tetrahedral meshing remains an important tool for computational simulation, experience has shown that complexity of the model does not affect the time to mesh as acutely for tetrahedral meshing as it does for hexahedral meshing. In addition, hexahedral meshes remain the mainstay of, and a necessity for, a majority of analysts at the US National Laboratories. Although some aspects of this work may be applicable for meshing CAD models in general, the principal focus is directed at the hexahedral meshing problem.

A new metric is proposed to estimate meshing complexity of a CAD part. An evaluation of the metric is performed through a study of generating unstructured hexahedral meshes for 24 CAD geometries. While little effort has been directly aimed at quantifying meshing complexity in the literature, a great deal of research has been performed to identify and remove geometry and topology errors in CAD models, for the purpose of mesh generation. This process is commonly referred to as *model clean-up*. The obvious relationship between model clean-up and meshing complexity is that geometry and topology errors often lead to increased model complexity and time spent on the meshing process. Various approaches for model clean-up are reviewed. Additionally, approaches and techniques for generating unstructured hexahedra are also reviewed.

1.1 Model clean-up

From a meshing perspective, the CAD model can typically have two kinds of problems: definition errors and representation problems. Both typically cause more time to be spent in the meshing process. Definition errors are those that deal with how the model is defined, both in geometry and topology. Representation problems can be more subjective and are dependent on how the model will be used.

1.1.1 CAD model repair

Computer aided design model repair is defined as the process of fixing geometric and topological definition errors in a design model. CAD models are represented by one of the two methods: boundary-representation or *b-rep*, and constructive solid geometry or *CSG*. Currently, most commercial vendors use the *b-rep*. *B-rep* models are represented by mathematical descriptions and lower- order boundary topology. For example, an *edge* is defined by a mathematical *curve* and is bounded by two *vertices* at the ends. The *curve* may be defined by a mathematical description including a simple line-segment, arc, or a more complicated B-spline. Similarly, a mathematical *surface*, and boundary edges define a *face*. *Volumes* are defined by a series of connected faces that wholly enclose a specific region.

Computer aided design model repair typically involves fixing the mathematical curve and surface definitions. Work in this area has focused on detecting errors within the CAD model, either directly in the native format (in the software it was created) or in a third party software package. There are many different errors which can be detected. These include: inverted faces, gaps between surfaces in a volume, folded geometry, surface geometry with no bounding face, faces with no finite area, self-intersecting edges and faces, face/edge sloppiness, boundary edges that do not lie on the faces, overlapping faces, etc. [1–3]. There are several vendors that offer packages that detect and fix these problems. Some of these include: ACIS 3D Toolkit [4], Parasolid BodyShop [5], and CADfix [6]. Another tool called CADIQ [7], connects directly with the major vendors of CAD software and interrogates models for errors. In environments where this package/software is used, the cost savings to analysts could be large since the majority of the time spent generating a mesh is used in iterating with the design engineer on a "good" base model [2].

Computer aided design Model Repair often appears unrelated to meshing since it is generally assumed at the meshing stage that the CAD models used for simulations are valid. Problems with the validity of the model are assumed to be taken care of upstream, or in the design stage. Unfortunately, designers have no incentive to validate their models for use in mesh generation since such models are typically created for visualization or manufacturing where model quality is less important [8]. In recent years, more meshing related software has been developed to fix these problems or build meshing algorithms that are less sensitive to geometry and topology definition errors [9, 10]. In either case, it has been clearly demonstrated that CAD Model Repair is part of the overall process in going from *design to analysis* (D2A) and should be counted as part of any system that evaluates model complexity with respect to meshing.

1.1.2 Model simplification

Model Simplification involves steps that are taken to detect and alter *representation* problems of a solid model to make mesh generation easier or possible. CAD models are typically built to accurately capture the detail of the real problem; a method which presents practical issues for discrete, numerical simulations. For example, Fig. 1 shows a part that has a filleted section "colliding" with the boundary of another face, producing a tangential intersection of the boundary edges. This tangential intersection is difficult to mesh at any realistic element size; but it is especially hard to mesh with quadrilateral elements. This tangency is not a result of how the CAD model is defined, but rather how it is *represented* with respect to its proposed use. Model

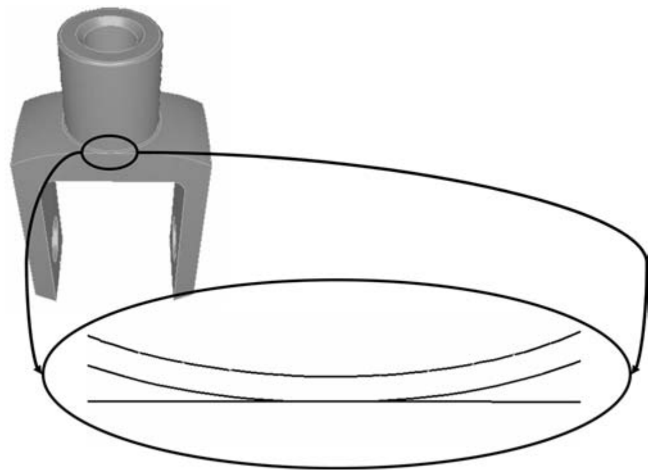


Fig. 1 Geometry problem due to a fillet

simplification is the process of detecting and removing such representation problems that make mesh generation difficult.

New approaches for simplifying CAD models for mesh generation have recently been proposed and successfully implemented [11–16]. These methods scan the model and search for specific preprogrammed problems, such as stray vertices, sliver surfaces, or small fillets. When the problems are found, they are either fixed in the native geometry subsystem or with a system-independent method. Sheffer et al. [11] first referred to this system-independent clean-up method as ''Virtual Topology''. Virtual topology provides a way to change the topological representation of the model by layering modifications on the model without changing the model itself. Many of the approaches mentioned [12–16] automate the removal of preprogrammed problems; however, this process is often subjective. It is, therefore, difficult to predict which problems can be removed to make meshing easier and which problems the mesh generators must handle for the purposes of analysis. For example, a fillet in a CAD model may be included to dissipate stress concentrations at a critical region or it may be merely cosmetic.

1.2 Unstructured hexahedral mesh generation

The study of meshing complexity is restricted here to generating unstructured hexahedra. Despite numerous efforts [17–24, 25, 26], there remains an absence of a satisfactory high quality automatic hexahedral meshing scheme. Instead, many researchers have attempted to improve the manual methods of generating hexahedral elements, namely sweeping and mapping. Some of these improvements include automating the traditional approaches by extending the sweeping and primitive algorithms and developing new automation control algorithms.

Mesh primitives are a set of pre-packaged meshes for typical or common shapes like squares, triangles, and circles in 2D and cubes, tetrahedrons, and spheres in 3D. Sweeping is essentially an extension of a cylinder primitive where the top circular surface mesh is extruded through the volume into hexahedrons. Sweeping requires that the ''linking'' surfaces or sidewalls of the sweep axis be meshed with a structured or regular meshing scheme like mapping [27].

In a typical manual approach, a user will decompose a part into pieces that can be meshed with either a primitive or sweeping algorithm. For instance, the model shown in Fig. 2a is not simply sweepable along the dominant axis of the part due to the protruding smaller cylinder on the side. In order to mesh this part, a sweep path must be cut through the larger material in order to sweep the side cylinder as shown in Fig. 2b. Additionally, the user must be careful while creating this sweep path not to interfere with the interior of the part. The final decomposition and mesh of this model is shown in Fig. 2c.

Decomposing all the parts into primitives or source-to-target sweeps is tedious and often unnecessary. Preprocessing steps to automatically perform a minimal amount of ''pseudo'' decomposition are added to extend sweeping and mapping. The mapping algorithm is extended to submapping, which uses virtual subdivision based on the boundary mesh to decompose a part into mappable sub-regions [28]. The final result in both 2D and 3D for the submapping algorithm remains a struc-

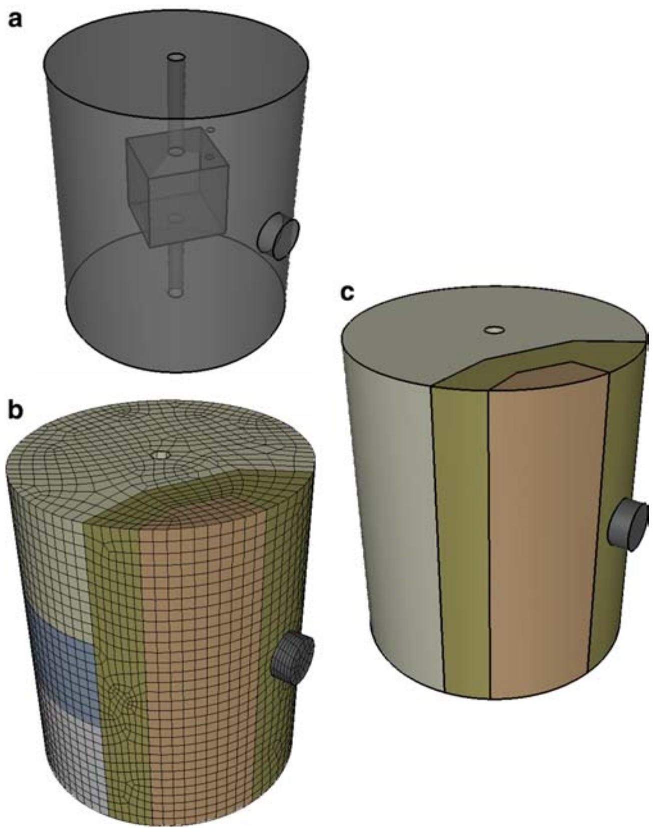


Fig. 2 Swept meshing approach coupled with decomposition

tured grid. Submapping in 2D is heavily relied upon to extend the sweeping algorithm by allowing more freedom with the linking surfaces. Sweeping is first extended to "pick-up" additional source faces in the sweep as it progresses through the axis. The algorithm is further extended to not only pick-up faces but also terminate them as the sweep travels along the axis [29]. This technique is referred to as "multi-sweep" or "Coopering".

Primitives and sweeping often rely on user intervention to prescribe exact boundary intervals, surface meshes, and sweep directions. When meshing large assemblies of parts, managing and entering this data can become overwhelming, even for experienced users. Automation for controlling and relaxing the amount of user-supplied data has been another area of research in hexahedral meshing. Two algorithms that have substantially reduced this problem are automatic scheme selection and automatic interval assignment. Automatic scheme selection uses a sweepability proof to detect shapes that can be meshed with sweeping and other primitives [30]. The algorithm automatically assigns the proper surface schemes and determines proper sweep directions. The automatic interval assignment algorithm solves a system of linear, integer constraint equations to provide proper edge intervals for meshing [31]. The constraints are based specifically on the requirements of the meshing algorithms that are to be used.

An approach to estimating the meshing complexity of various CAD geometries is presented in this paper. The resulting information is intended to provide managers and users of meshing software a method of predicting how difficult a CAD model will be to mesh, enabling them to make decisions about model simplification and analysis approaches prior to meshing. The developments here pertain to individual parts in an assembly. Further developments would necessarily include the consideration of complexity of the entire assembly.

The complexity of meshing a CAD model is quantified using a metric. The components of the complexity metric discussed in this paper are intended to aid users in identifying features in the geometry that make mesh generation difficult. Such information will further aid in deciding if these features are necessary and what effect their removal will have on the meshing process.

2 Meshing complexity

Meshing complexity is a measure of the level of difficulty encountered in meshing a CAD geometry. Several aspects are known to make mesh generation difficult including near- tangencies, topology arrangement, etc. Meshing complexity involves the translation of these difficulties into a metric that quantifies them. For instance, there are many geometries that have extremely geometrically complex curves and surfaces but are meshed trivially. Likewise, one could easily construct a model using the linear curves and planar surfaces that is nearly impossible to mesh. The problem posed by meshing complexity is finding what actually makes meshing difficult. Since many of these factors may be non-quantifiable, a reproducible and all-encompassing solution may be intractable. Instead, a solution is proposed that focuses on a subset of quantifiable issues to develop a useful measure for mesh complexity.

2.1 Variables of meshing complexity

Finding a single metric that accurately captures the meshing complexity of a solid is a complex problem. The problem can be viewed as determining a function that evaluates the complexity of a solid with respect to meshing. In developing such a function, the following variables may be considered: *element type*, tet versus hex ; *element structure*, structured versus unstructured; *element size*, coarse versus fine; *topology; geometry; assembly configuration*, multiple solids; *finite element analysis (FEA) application*, e.g. boundary layers; *user expertise; meshing software maturity;* and *choice of meshing algorithm*. To reduce the scope of the problem, *meshing software maturity* and *user expertise* should naturally be removed because both are highly unpredictable.

It is assumed here that models will be meshed with unstructured hexahedra for structural mechanics applications, thus fixing the variables of *element type, element structure*, and *finite element analysis application*. Also, by choosing unstructured hexahedra and in the absence of an automatic algorithm, the *choice of meshing algorithm* variable will be assumed to be the methods discussed in Sect. 1.2, namely sweeping and mapped mesh generation (including the multi-sweep and submapping algorithms). Further, assembly models are not considered. With these assumptions and restrictions, this paper seeks an approximation of the meshing complexity function with the following variables: *element size, topology, and geometry*.

Even when limiting the scope of the proposed mesh complexity metric to these factors, it should be noted that by its very nature, the values that go into defining the metric are somewhat heuristic. Extensive experience with meshing complex models has led to the emergence of the proposed complexity metric. Given the same set of circumstances, another individual may develop the metric in a different manner; however, the principles proposed in this work are universal.

2.2 Meshing complexity function

The chief problem for unstructured hexahedral mesh generation lies in decomposing the model into suitable parts that are meshable with two available algorithms, namely: multi-sweep and submapping. It is, therefore, desirable for the approximation function to capture the shape of the part and to determine whether or not it can be swept. Variables that contribute to the shape of the model are termed *base metrics*. The approximation function should also consider those features in a model that make it difficult to mesh, such as topology and geometry problems like small curves, sliver surfaces, and small angles. Variables that are detrimental to meshing the part are termed *negative metrics*. There are various ways to empirically combine the base and negative metrics to compute mesh complexity (C). The following is proposed for such a combination:

$$C = \frac{\sum_i = 1^n w_i B_i}{\left(1 + \sum_{j=1}^{k} N_j\right)^2}, \tag{1}$$

where C is the new meshing complexity approximation function; $0 \leq C \leq 1$; n is the number of base metrics; B_i is the base metric, i; $0 \leq B_i \leq 1$; N_j is the negative metric, j; and w_i is the base weight, i, with

$$\sum_{i=1}^{n} w_i \leqslant 1 \tag{2}$$

$C = 1$ for geometries that are trivial to mesh and vanishes for shapes that are challenging. It should be noted that Eq. 1 does not assume any specific factors or number of factors for N_i. As such, the complexity equation can easily be augmented to suit new negative metrics as they are determined to be useful. For the purposes of this study, and from extensive experience, we have delineated a specific set of base and negative metrics on which we will develop our results.

The following base metrics are used: *inverse topology count*, *sweep detection*, and *cartesian edges*. The negative metrics considered most effective are the number of: *small curves, small surfaces, close loops*, *small and large angles, bad CAD definitions*, *groups of blend faces*, and *tangential surface intersections*. The one exception to Eq. (1) is where the sweep detection metric has a value of unity. If this occurs, then C is set equal to unity, as will be explained in Sect. 3.1.2.

For the purpose of this research, a software program, SEER, was implemented to compute the meshing complexity metric. SEER uses the common geometry module (CGM) [32] for its geometry query and model representation, and currently uses ACIS as the underlying geometric core. For input, SEER requires a solid model in the ACIS SAT, IGES or STEP formats. The user of the software must supply the solid model to be measured and the desired element size.

3 Computation of metric

The meshing complexity metric is computed by examining the CAD model and measuring the base and negative metrics of the meshing complexity function.

3.1 Base variables

In the proposed metric, three base metrics are included, namely, *inverse topology count*, *sweep detection*, and *cartesian edges*. The purpose of base variables is to provide a starting range for the metric. Many parts that are difficult to mesh do not display any *negative* aspects. The base variables must, therefore, by themselves calculate the difficulty of the shape with respect to meshing. These three base variables do not capture the complexity entirely, but do offer an adequate beginning for most models.

3.1.1 Inverse topology count

The inverse topology count variable, I, is defined as:

$$I = \frac{1}{2}\left(\frac{6}{F} + \frac{12}{E}\right) \text{ while } \frac{6}{F} + \frac{12}{E} \leq 2, \text{ otherwise } I = 1 \tag{3}$$

where F is the number of faces in the model and E is the number of edges. The scalar numerators ensure that the variable equates to unity for a cube. The restriction of limiting the metric to unity comes from shapes that have fewer faces and edges than that on a cube, like a sphere that has one face and zero edges.

The inverse topology count variable arises from the observation that as the number of topological entities increase, the difficulty of meshing the object also increases. If this were strictly the case, the inverse topology count variable would be the only variable needed. This, of course, is inaccurate since many cases have numerous faces and edges but are easy to mesh. Likewise, there are cases where there are relatively few faces and edges that are difficult to mesh. However, as a rule, the inverse topology count generally reflects the difficulty of meshing directly and as such is considered a relevant base variable.

3.1.2 Sweep detection

Sweep detection directly addresses the major goal of the meshing complexity function: to identify models that are sweepable or to determine how easy it would be to transform the models into sweepable pieces. The sweep detection variable is based on the ideas proposed by White and Tautges [30]. This method, called *auto sweep detection*, connects the linking surfaces and traverses them in the opposite direction to determine sweep direction and source/target face identification. The method, however, does not differentiate between parts that are almost sweepable and parts that are not. The sweep detection variable proposed here uses the auto sweep detection method with some modifications to determine how "close" to sweepable a part would be.

Pseudo code for the algorithm to determine the sweep detection variable is given in Algorithm Table 1. In step 2 of Algorithm Table 1, the auto sweep detection algo-

Table 1 Sweep detection metric algorithm

```
1. Let $V$ be the CAD part being measured
2. If $V$ is sweepable (use auto sweep) THEN
3. LET $C = 1.0$;
4. ELSE
5. If V is a primitive shape, THEN
6. LET $C = 0.95$;
7. ELSE
8. LET $C = \zeta(V)$, where $\zeta$ is the partial sweep detection metric
9. RETURN $C$:
```

rithm is called. The auto sweep detection algorithm consists of four procedures to determine if a part is sweepable.

The first procedure is to classify the Cartesian traversal types of the interior vertex angles for each surface and set the surface meshing schemes. At each vertex on every surface the interior angle is calculated by measuring the angle between the two edges that join at that vertex. For non-linear edges, the tangent of the curve where it hits the vertex is used. The Cartesian traversal type, or vertex type, is assigned by rounding the angle to the closest Cartesian angle of 90, 180, 270 or 360°. Based on the Cartesian angles the following values are assigned to the vertices for each surface (a vertex can have more than one vertex type value since it can be used differently for each surface of which it is a part): 1 for 90°, 0 for 180°, −1 for 270° and −2 for 360°. After all the vertices on a surface have been assigned, the meshing scheme of the surface is assigned based on the vertex types. If the sum of the vertex types on the surface is equal to four, then the surface is submappable. If the sum is not equal to four, then the meshing scheme is set to an automatic unstructured scheme, such as Paving [33] or Q-Morph [34].

The second procedure is to find *chains* or complete loops of submappable surfaces. Submappable surfaces have a logical 2D parameter space, *i*–*j*. The edges of the surfaces can be classified into four different boundaries of the parameter space: $+i$, $-i$, $+j$, and $-j$, where $-i$ is opposite $+i$ and $-j$ is opposite $+j$. When submappable surfaces are connected with common edges, the parameter space can continue through to adjacent surfaces following the opposite parametric sides. Figure 3 shows an example of how the submappable surfaces are connected to form loops or chains of surfaces connected through opposite sides in the parameter space. The arrows in the Figure are drawn to indicate the direction of the sweep chains. For this example four chains are visible with one additional chain hidden in the hole. The chains are considered complete if and only if they are non-self intersecting, meaning that the chain cannot begin in the "*i*" direction on a surface and then later cross the same surface in the "*j*" direction. Each edge in the direction maintained by the chain must also be connected to a submappable surface, or in other words, the chain must be complete or fully wrapping. Presence of these chains is a prerequisite for the volume to be swept [30].

In the third procedure, the edges of the volume are classified into Cartesian or edge types similar to the classification of the vertices for the faces. The surface normals on each side of every edge are used to measure the dihedral angles between the two faces connected at that edge. The dihedral angle is rounded to the nearest Cartesian angle and classified to the values of: 1 for 90 degrees, 0 for 180 degrees, −1 for 270 degrees and −2 for 360 degrees.

Finally, the fourth procedure is to traverse the chains found in the second procedure. To start, a face is chosen that is either not submappable or if all surfaces are submappable then an arbitrary face is taken and set to be the initial source face for the sweep. Each of the faces of the volume are reached by traversing from the edges of the first face, then recursively the edges of the next faces are found in a depth first search. The source and target faces are found by ensuring that between each source or target face there is a complete chain that runs in the direction opposite to the traversal, and that there are non-zero edge types between the source/target surfaces and the chains. For example, in Fig. 3 let the initial face be Face 10. The edge between Face 10 and Face 9 has an edge type of 1. Face 9 is also in a complete chain and the edge between Faces 9 and 10 is not part of the chain. The algorithm would then move from Face 9 to Face 8. The common edge between Faces 8 and 9 has an edge type of −1, indicating that Face 8 should also be a source face with Face 10. The search would continue across Faces 6 and 5 until Face 11 (underneath) would be found and identified as a target face because of the edge type. This process would continue until all the faces are traversed, resulting in Faces 10, 8, 4 and 2 being selected as source faces and Face 11 as the target face.

After auto sweep detection is run, as indicated in Algorithm Table 1, if the CAD part is not determined to be sweepable, the part is tested in step 5 to see if it is one

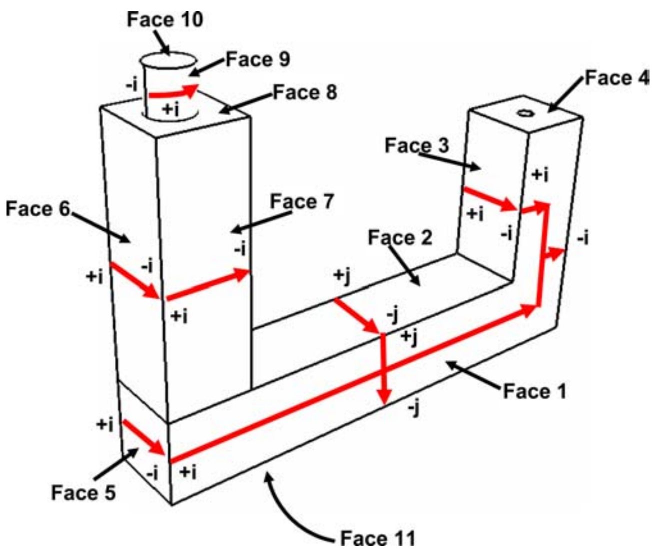


Fig. 3 Sweeping chains

of the several primitives such as: sphere, half sphere, torus, tetrahedron, or cone.

If the part is neither a primitive nor sweepable, the function $\zeta\ (V)$, or the partial sweep detection metric, is used to determine the sweep detection variable. The pseudo code for $\zeta\ (V)$ is given in Algorithm Table 2. Step 6 of Algorithm Table 2 is picking the corners of the faces that are "forced" to be represented with the rectangular meshing primitive. Corner picking, devised by Mitchell [35], is applied to choose the most appropriate corners of the face.

In step 7, the third procedure of the auto sweep detection algorithm is redone; namely chains of linking faces are determined. If no chains are found, the sweep detection metric is set to zero in step 9. Otherwise, in step 11, the chains are used to compute the sweep detection metric S. The sweep detection metric is computed as:

$$S = M_\mathrm{b} * M_\mathrm{h} * M_\mathrm{c} * MAX\left(\frac{TCF}{q}, \frac{\sum_{\mathrm{i}=1}^{\mathrm{m}}\sum_{\mathrm{j}=1}^{\mathrm{l}} A_{ij}}{\sum_{\mathrm{k}=1}^{\mathrm{q}} A_k}\right), \qquad (4)$$

where M_b, M_h, and M_c are modifiers based, respectively, on how much the chains cover the volume, the number of cylinder holes, and the large numbers of chains present; MAX is the function to return the maximum of two scalars; TCF is the number of faces in the chains; q is the number of faces in the volume; m is the number of chains; l is the number of faces in chain i; A_{ij} is the area of the face j in the chain i; and A_k is the area of the face k in the volume.

The modifier M_b is determined by comparing the union of all the bounding boxes of the chains and the bounding box of the volume. The comparison is made by checking the number of directions out of three (x, y, and z) in which the bounding box of the chains is equal in size to the bounding box of the volume. For example, if the bounding box for the chains is {{0,0,0}, {1,1,1}} and the bounding box of the volume is {{0,−1,0},{1,1,1}} then the boxes will be equal in two directions. The modifier, M_b, is equal to 0.9 if the boxes are equivalent in no directions, 1.4 if there is one direction, 2.0 if there are two directions, and 4.0 if the boxes are equivalent in all three directions. These values were determined through numerical experiment.

The modifier M_h is determined by finding the number of chains that are formed by simple cylinder holes with one or two surfaces constituting the entire chain. In general, these types of chains do little to determine how sweepable the part is, so this circumstance is viewed as a reducing modifier. M_h is computed by finding the fraction of chains that are not cylinder holes out of the total number of chains. This modifier is lower bounded to be 0.01 since the presence of holes indicates some form of a sweep direction.

The final modifier M_c is another reduction factor to reflect whether there are numerous chains on the volume. If there are more than ten chains, their presence can indicate that the volume has many sweep directions, meaning that meshing may be more difficult than volumes that have relatively few sweep directions. Therefore, if the bounding box of the chains is not equivalent in all three directions to the bounding box of the volume, and there are more than ten chains, M_c is set to be equal to the inverse of the number of chains; otherwise, M_c is unity.

There are, of course, many counter examples for the sweep detection metric, which is why it is only a part of the entire meshing complexity metric. However, as a general rule, the metric is shown to yield lower values for geometries that are more difficult to mesh and correctly predicts higher values for trivial geometries.

Table 2 Partial sweep detection metric algorithm

```
1. FOR EACH fave f in v
2. LET X (f) be the mesh scheme of f
3. LET Y(f) be the number of holes in f
4. If X(f) is unstructured AND y(f) is > 1 THEN
5. LET X(f) = Rectangle primitive
6. Choose the best corners for f
7. LET LC be the number of linking chains in v
8.If LC = = 0 THEN
9. RETURN 0.0
10. ELSE
11. return S(V)
```

3.1.3 *Cartesian edges*

The *Cartesian edges* metric is intended to capture the degree to which the volume is of a "blocky" nature. In general, a blocky volume is easier to mesh since the hexahedra fit easier when the faces are aligned orthogonally. The metric is computed by first counting the number of edges connected to planar surfaces with Cartesian dihedral angles between them. For this metric, Cartesian dihedral angles are specified to be 90, 180 and 270°, respectively. Dihedral angles that fall within three degrees of these values are considered Cartesian. After the number of Cartesian edges is found, the metric is calculated by dividing this number by the total number of edges in the volume. The Cartesian Edges ratio is found to be inaccurate and arbitrary when it is less than 0.4. Therefore, the metric is set to zero for volumes with ratios of less than 0.4, and set to the ratio itself for ratios greater than 0.4.

3.1.4 *Base metric weights*

The weights for the three base metrics were computed by trial and error. It was found that 0.1 would be the best weight for the *Inverse topology count* metric. For the *Sweep detection* metric, the best weight was experimentally found to be 0.5. Again by numerical experiments, the *Cartesian edges* metric was given a "stepping" weight based on the metric itself. The Cartesian edges weights are defined in Algorithm Table 3. It is acknowledged that if the Cartesian Edges metric is less than 0.6 then the sum of the weights will be less than one, as reflected

Table 3 Cartesian edge metric weight calculation

```
1. LET $CE$ be the computer cartesian Edge metric value
2. LET $w_{CE} = 0.0$ where $w_{CE}$ is the weight of the cartestian Edge
   metric
3. If $CE >= 0.4$ AND $CE < 0.5$ THEN
4. LET $w_{CE} = 0.1$
5. ELSE IF $CE >= 0.5$ AND $CE < 0.6$ THEN
6. LET $w_{CE} = 0.2$
7. ELSE IF $CE >= 0.6$ THEN
8. LET $w_{CE} = 0.6$
```

in Eq. (2). It was, however, necessary to reflect the increased importance of the Cartesian Edges metric as the value of the metric increased.

3.2 Negative metrics

The negative metrics are used to cause a decrease in the value of the meshing complexity metric. The negative metrics reflect problems found in the model that will be detrimental to mesh generation. This study was unable to determine a ranking of these problems in terms of their impact on the meshing process, and as such they are weighted equally at this time. Additionally, test results revealed that all the negative metrics were rarely found in a single part. This indicates that meshing complexity can vary greatly between parts, and that the list of negative metrics mentioned here may be incomplete. New negative metrics may be added to the existing list until a general metric is found.

The negative metrics identified during this study are now presented. Unlike the base metrics, these are all determined by counting the number of occurrences of a given problem. Additionally, many of these metrics are functions of element size, meaning that if the desired element size for meshing the volume changes it could impact the results of these metrics.

3.2.1 Number of small edges, faces and close face loops

Small edges, faces, and close face loops all adversely affect the meshing process, making it difficult and sometimes impossible to generate a reasonable quality mesh. In fact, such artifacts often lead to robustness issues with automatic surface meshing algorithms unless mesh sizing in the region is performed carefully. Finding these entities visually is often difficult. Additionally, before the final mesh is achieved, these entities must usually be removed from the model, adding additional time to the mesh generation process.

The number of *small edges* is equal to the number of edges in the model with a length less than or equal to 1/5 the element size given as input.

The number of *small faces* is equal to the number of faces with a hydraulic radius less than or equal to 1/5 the element size. The hydraulic radius [15] is computed as,

$$r_{\mathrm{h}} = 4A/P,$$

where A is the area of the face and P is the total length of the perimeter of the face.

The number *of close loops* metric is equal to the number of minimum distances between loops that are less than or equal to 1/5 the element size.

3.2.2 Number of small and large angles

Small and large angles can also affect the quality of the mesh. When the angles are very small, the geometry must be modified in order to generate a suitable mesh. The angles for this metric are calculated at vertices on a surface and curves on a volume. For vertices the angle is calculated by measuring the angle between the tangents of the two edges at that vertex on a particular surface. For edges the angle is calculated by measuring the angle between the normal vectors of the two faces that share the common edge. The normals are measured at the midpoint of the edge and assumed to be constant throughout the length of the edge. The number of small and large angles is computed by counting all the vertex and edge angles that are obtained by measuring all the vertex and edge angles that are smaller than 20° or larger than 340°.

3.2.3 Number of bad CAD definitions

The ACIS geometry engine [4] is used in this study to represent the underlying geometric definitions of the CAD models. As mentioned in Sect. 1.1.1, for various reasons these definitions may be inaccurate or faulty. Many of these problems can be fixed via the healing technologies available in various software packages. In general, the presence of these problems indicates an increased time to generate a mesh. If the problems can not be fixed, meshing can be difficult since these problems can limit the use of decomposition tools at one extreme and make meshing impossible without rebuilding the geometry at the other. The ACIS geometry engine provides the ability to detect geometry definitions errors. For computing this metric, the ACIS software is queried directly to determine how many entities have problems.

3.2.4 Number of groups of blend faces

Blend faces, more commonly referred to as fillets and rounds, are commonplace in CAD models. Their presence is usually directly related to increased meshing times, especially for hexahedral meshing. Cartesian edges are typically best for mesh quality in hexahedral meshing. Additionally, presence of blend faces usually does two things to a model: add extra topology that hinders mesh generation, and remove topology that is needed for sweeping. An example of this is shown in Fig. 4 where a brick of dimension 10 units has all of its curves blended with a radius of 1.0. The result is that this trivial meshing shape becomes more difficult and requires topological modifications before it can be meshed.

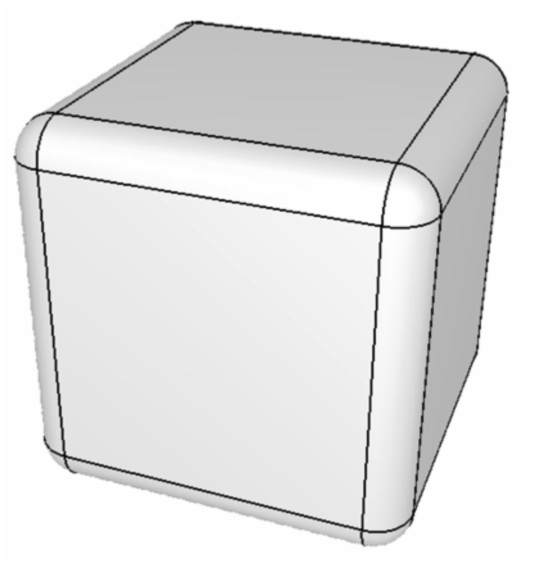


Fig. 4 Brick with all edges blended

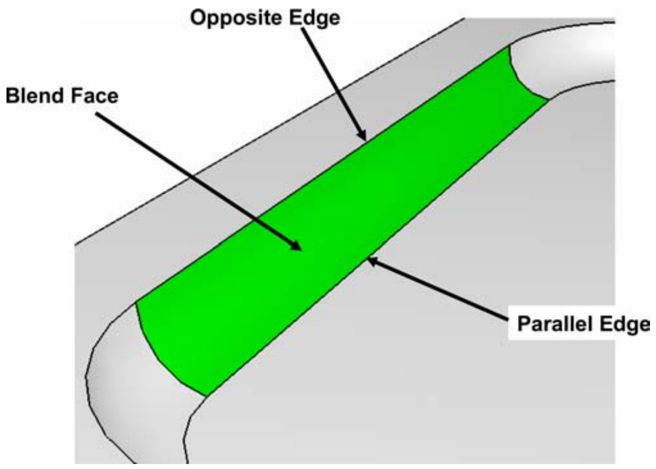


Fig. 5 Blend face

The best approach for dealing with these entities for hexahedral meshing is to remove them and return the model to Cartesian intersections. In many cases this is not allowable since, while some of the blends are present for appearance, many more are there for physical reasons. Regardless of whether or not the blends can be removed, their presence leads to mesh generation challenges and increases the time required to generate a mesh, making it important for the meshing complexity metric to capture these faces.

A *blend group* is a set of connected blend faces that are generally blending the same area although the edge they blend may change for various reasons. Because of these changes, the number of blend groups is found rather than the number of blend faces. While others have attempted to detect blend faces [15], a new method is presented here.

For blend face detection, the following three rules are made. First, blend faces have at least one edge where the faces attached to that edge have a dihedral angle of 180° at the edge. This edge is called the *parallel edge*. Second, an edge opposite the parallel edge has a Cartesian angle, 90, 180, 270°, between the two faces that are attached to it. The edge opposite the parallel edge is called the *opposite edge*. And third, the angle between the face normals on the parallel edge and the opposite edge is also Cartesian. An example of a blend face that fits these three rules is shown in Fig. 5 where the parallel and opposite edges are identified.

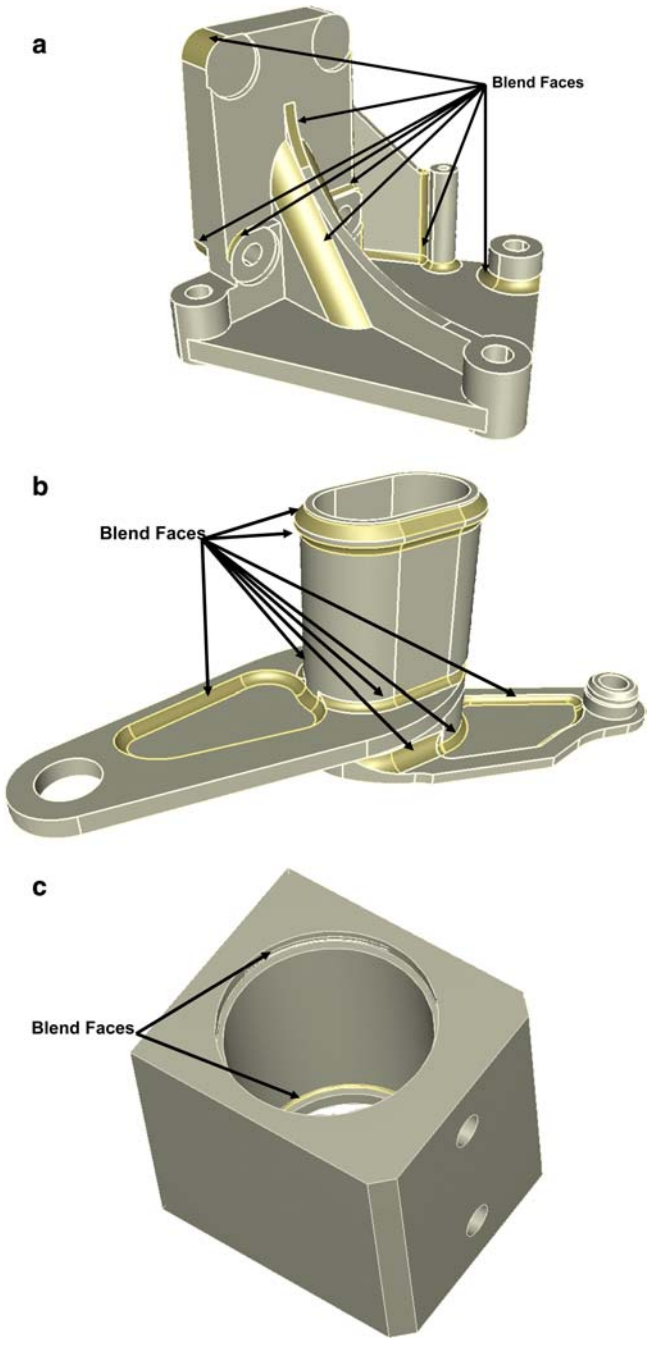


Fig. 6 Blend detection on test parts 1(**a**), 20(**b**) and 5(**c**)

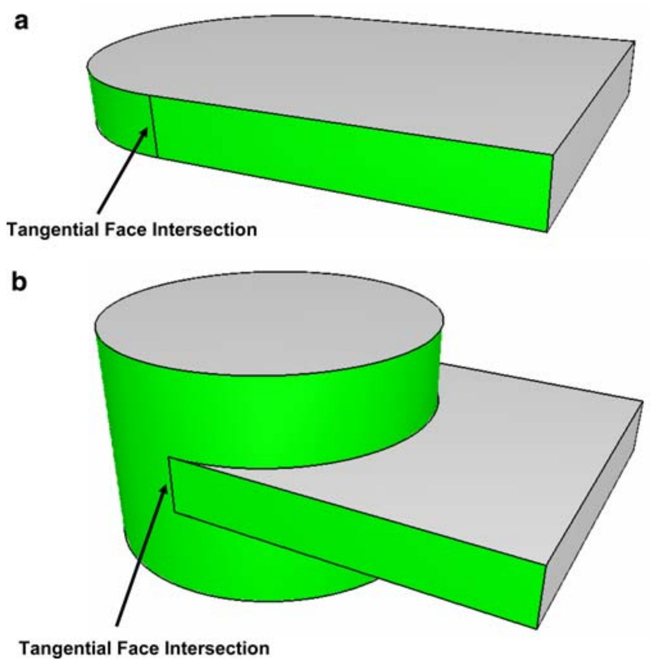


Fig. 7 Tangential face intersections

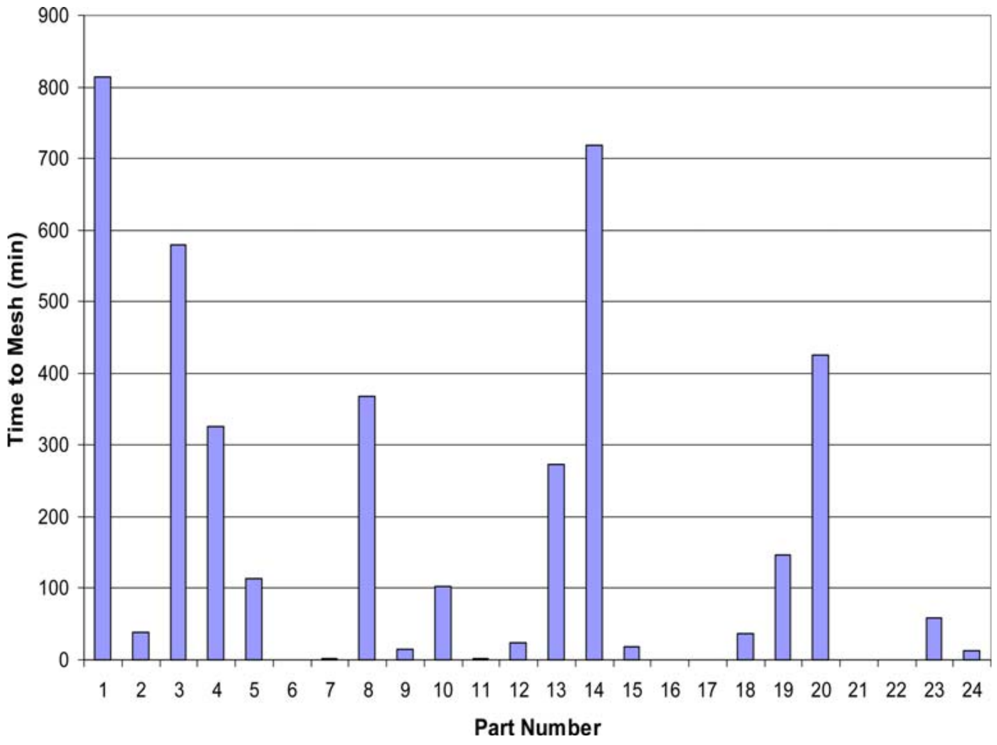
Table 4 Test suite part numbers and corresponding time to mesh

Part number	Element size	Time to mesh (min.)
1	2	813
2	8	39
3	1	579
4	10	327
5	0.1	114
6	4	0.17
7	1	2
8	5	369
9	2	15
10	0.5	102
11	0.005	2.5
12	3.5	23
13	7	273
14	2	719
15	0.2	18
16	0.5	0.083
17	23.9	0.083
18	2	36
19	2	146
20	0.009	425.5
21	0.5	0.017
22	10	0.083
23	2	58
24	5	13.5

To find blend faces, the model is searched for faces that meet these criteria. Determining the opposite edge is the most difficult part. The algorithm starts with the parallel edge, then searches the other edges on the face for the edge that is at the closest distance and most parallel to it. The search is done by traversing the edges counter-clockwise around the face. The edges are not tested until a vertex angle of less than 135° is passed. The edge selected in this process is called the *opposite edge*. If the face only has three curves, then the face is a blend face only if all the tested curves meet the first rule, and all edges are attached to other blend faces.

After all the blend surfaces have been detected, the blend surfaces themselves are traversed going from their edges to adjacent faces using a union-find algorithm to group the blend faces that are connected by an edge. The

Fig. 8 Test parts versus the time to mesh



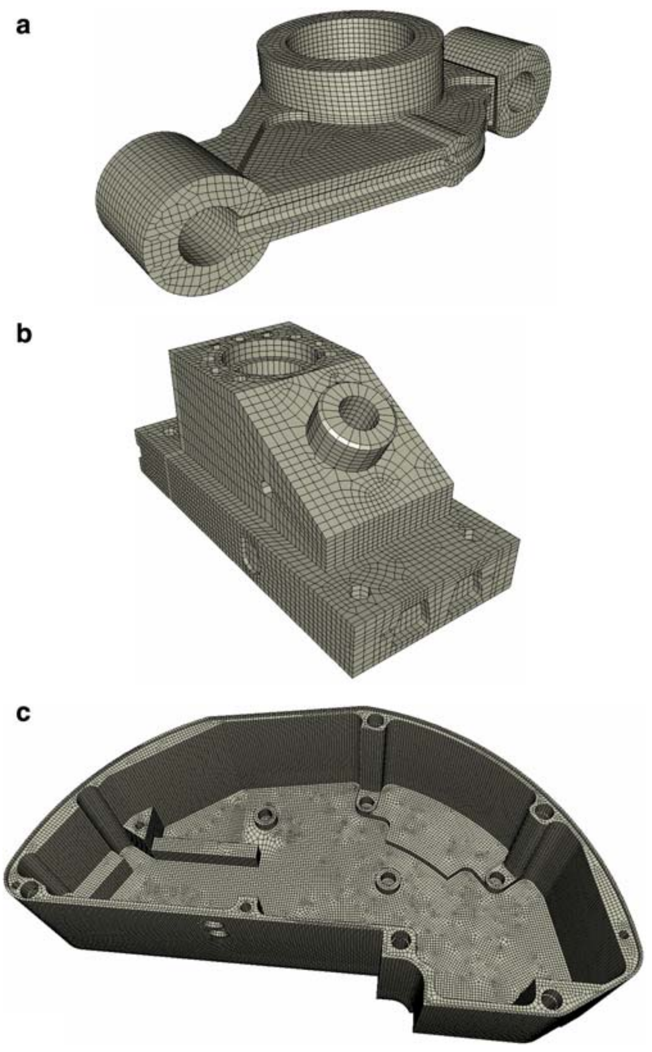


Fig. 9 Test parts 3 (**a**), 13 (**b**) and 14 (**c**)

groups of blend faces metric is calculated by counting up the total number of blend groups found from this algorithm. Figure 6a, b and c are test parts with blend faces. The blend algorithm is able to find all the blend faces in these models. Several faces that would typically not be considered blend faces by visual inspection are also included as a side effect of the algorithm. In general, the algorithm is able to detect blend surfaces, and especially the hidden ones shown in Figure 6c that make mesh generation difficult.

3.2.5 Number of tangential face intersections

The final metric included in the negative metrics is the number of *tangential face intersections*. This metric seeks to find dissimilar geometrical faces that intersect tangentially and cause problems with mesh generation. Having two faces intersect tangentially is generally not a problem for mesh generation. Figure 7a shows an example of two faces that intersect tangentially that offer no resistance to mesh generation. In this figure, the two faces intersect without creating any small features. Figure 7b, however, shows an example of the tangency causing small features. Mesh generation is difficult when tangential face intersections are ‘‘capped’’ by a face with small vertex angles at the intersection. These configurations are identified in this metric.

Tangential face intersections that cause problems are identified by the following 4 steps. First, the user must gather all the faces and vertices that have small vertex angles. This information can be reused from the computation of the *small and large angles* metric discussed in Sect. 3.2.2. Second, identify the face that makes a 90° dihedral angle with the face with the small vertex angle. This face will be attached to either of the two edges that meet at the vertex with the small angle. Third, on the face found in step two, find the edge that is also connected to the vertex with the small angle but is not attached to the face where the small angle is. And fourth, measure the dihedral angle for this edge to ensure that this edge has a tangential intersection. The angle should be 180°. If all of these steps are done successfully, a tangential face intersection is recorded and counted. The metric is computed by counting all such instances.

4 Results

The meshing complexity metric is best evaluated on real CAD geometries rather than contrived test parts. A set of 24 CAD models were obtained for this purpose. The models range from simplistic to challenging in terms of their difficulty with respect to mesh generation. This section will discuss evaluation of the twenty-four models and the resulting meshing complexity metrics.

4.1 Test suite

To test the validity of the meshing complexity metric, CAD models were obtained and meshed with hexahedral elements. Twenty-four test models were obtained from various industrial partners. The meshing was done with the CUBIT software package [36] that contains the meshing algorithms described in Sect. 1.2. One of the authors performed all of the meshing so that there would be no differences in the time taken to mesh the parts based on differences in user expertise. All the time taken to mesh the models was recorded, including time spent thinking about approaches, trying different approaches, and time spent recovering from user mistakes. This is important to note since the meshing complexity metric must try to capture in some form which geometries and topologies cause the user more time. Time wasted due to bugs in the meshing software was discarded because of the assumptions stated in Sect. 2.1. Table 4 shows the part numbers, the element size for which they were meshed, and the total time taken to

Table 5 Base metric values for test suite

Part number	Time to mesh	Inverse topology count	Sweep detection	Cartesian edges
1	813	0.044323607	0.021336	0
2	39	0.028339818	0.8	0
3	579	0.089935065	0.000964	0
4	327	0.061111111	0.038482	0
5	114	0.1875	0.56875	0.419355
6	0.17	0.146320346	1	0
7	2	1	0.95	0
8	369	0.052859421	0.08	0
9	15	0.17375	0.545189	0
10	102	0.203917051	0.428571	0
11	2.5	0.348484848	1	0
12	23	0.136090226	0.000237	0.75
13	273	0.059738458	0.026286	0
14	719	0.019699932	0.011189	0
15	18	0.070211039	0.8	0
16	0.083	1	1	1
17	0.083	1	1	0.4
18	36	0.158004158	0.486486	0
19	146	0.049500958	0.041284	0
20	425.5	0.049331551	0.030319	0
21	0.017	0.529411765	1	0
22	0.083	1	1	0
23	58	0.055927835	0.031253	0.497908
24	13.5	0.366666667	0.8	0.571429

Table 6 Negative metric values for test suite

Part number	Time to mesh	Small edge, face, loop	Bad angles	Bad geometric def.	Blend groups	Tangential meetings
1	813	2	17	9	23	30
2	39	0	0	0	4	0
3	579	5	8	2	5	16
4	327	10	2	6	5	0
5	114	4	4	0	3	0
6	0.17	0	0	0	0	0
7	2	1	0	0	0	0
8	369	6	1	11	12	2
9	15	4	2	0	1	0
10	102	2	12	0	0	8
11	2.5	1	0	0	0	0
12	23	1	0	0	0	0
13	273	6	2	0	11	4
14	719	42	15	0	50	30
15	18	13	0	0	0	0
16	0.083	0	0	0	0	0
17	0.083	0	0	0	0	0
18	36	1	0	0	3	0
19	146	0	2	0	13	0
20	425.5	3	13	0	11	10
21	0.017	0	0	0	0	0
22	0.083	0	0	0	0	0
23	58	0	0	0	18	0
24	13.5	0	0	0	0	0

generate the mesh. Figure 8 shows a plot of part numbers and the times taken to generate the mesh in order to visualize the variations of the models. The time to mesh varies from under a minute to 833 min for the part that took the longest. Three of the parts and resulting meshes are shown in Fig. 9. Additionally, the parts shown in Fig. 6a–c are part numbers 1, 20 and 5, respectively.

4.2 Correlation of Data

The proposed meshing complexity metric is compared with times taken to mesh each part in the test suite. The results of this experiment are shown in Fig. 10; where, time to mesh is on the x-axis while the mesh metric is plotted on the y-axis on the logarithmic scale. The logarithmic scale is used to plot this axis since the mesh metric asymptotically approaches zero as the parts become more complex. The graph shows that the meshing complexity metric predicts the time to mesh within some range of error. Specifically, it shows that as the time required to mesh the parts increases, the metric value decreases asymptotically towards zero. For cases where the metric does not correlate well, this could be due to both the limitations of the new metric and the difficulty in prediction of human computer interactions.

Table 5 shows the results of the individual base metrics for the 24 test cases. For most models, the dominating base metric is the *Sweep detection* metric. Figure 11 shows a plot of the time to mesh each of the test cases against the corresponding *Sweep detection* values. The graph indicates that the lower *Sweep detection* values correspond to shapes that took longer to mesh. The graph, however, also shows several exceptions to this trend.

Table 6 shows the results of the individual negative metrics with the corresponding meshing times and part numbers. This table shows that the models that took longer to mesh contained more negative features, indicating the adverse affect these features have on the meshing process.

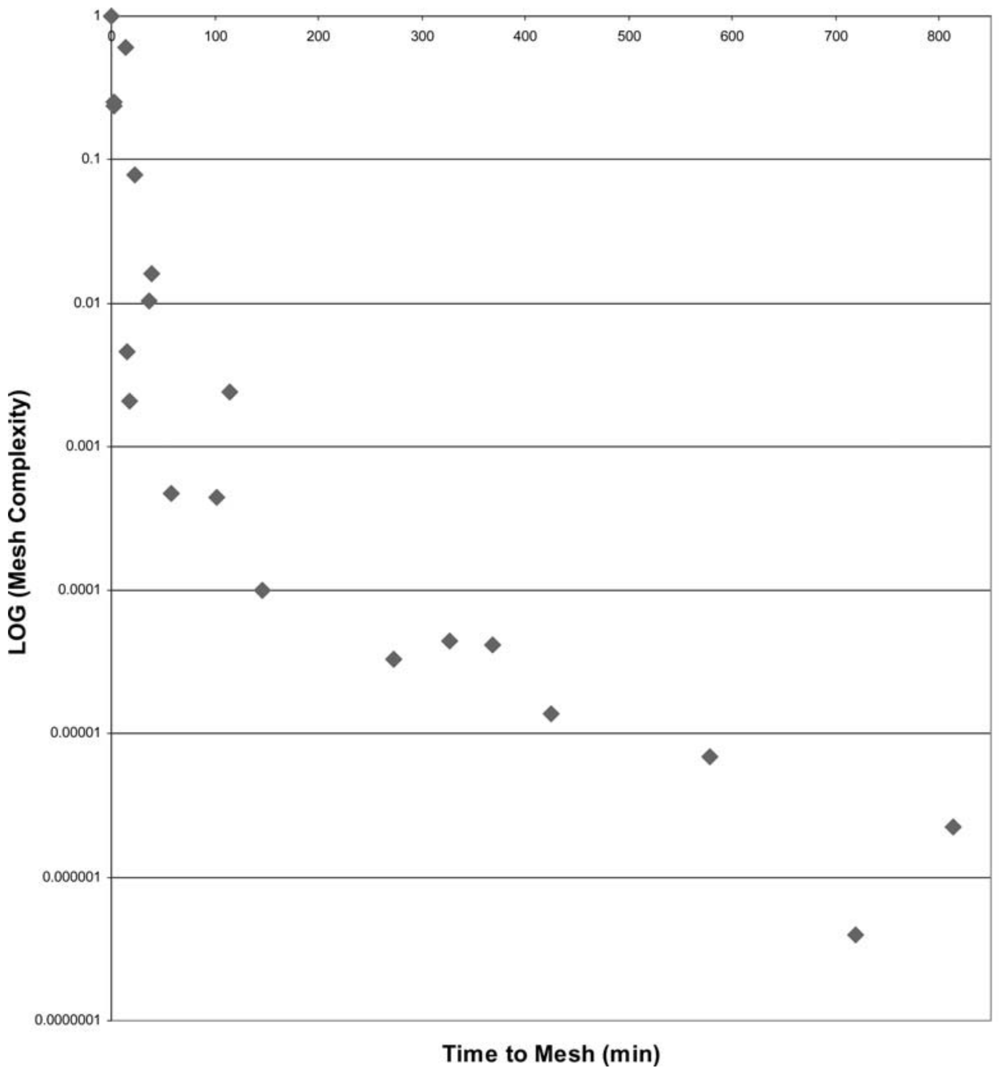


Fig. 10 Meshing complexity metric versus time to mesh

Using the data in Tables 5 and 6, the weights of the base metrics in Sect. 3.1.4, and Eq. 1, the meshing complexity metric results in Fig. 10 can be computed. To demonstrate this, the meshing complexity metric is calculated for part 1. The values for various parameters of Equation 1 are obtained from the first row of Tables 5 and 6 as: $B_1 = 0.044323607$, $w_1 = .1$, $B_2 = 0.021336$, $w_2 = 0.5$, $B_3 = 0$, $N_1 = 2$, $N_2 = 17$, $N_3 = 9$, $N_4 = 23$ and $N_5 = 30$. After inserting these values into Eq. 1, the meshing complexity C for part 1 is 2.24574×10^{-6}, as indicated in Fig. 10.

Several improvements could be introduced to the meshing complexity metric. First, the metric lacks a general base metric, which determines how sweepable the part is. While the sweep detection metric attempts this, it is not always accurate and can give incorrect answers at times. If a general base metric is found, it would also most likely lead to automatic decomposition approaches to vastly reduce the time to mesh for hexahedral elements.

Second, not all variables that affect meshing complexity are accounted for in this study. For example, in Sect. 2.1, it was assumed that *user expertise* would not be considered a factor. In theory, this should be the case if the metric compares parts that were meshed by the same user. In practice, however, this may not be true as human interaction is difficult to predict. For example, on some of the parts in the test suite the solution to meshing the part was immediately recognized, leading to a shorter time to generate the mesh. Other problems required more thinking, and time spent reflected this trial and error. To improve the metric, the human computer interactions aspect of the problem could be incorporated. Despite these inaccuracies, the proposed metric does predict the relative difficulty between the parts in terms of the time required to generate a valid finite element mesh within a range of error.

5 Conclusion

A new metric is presented to predict the meshing complexity of CAD parts. Meshing complexity refers to the relative difficulty to generate an unstructured hexahedral

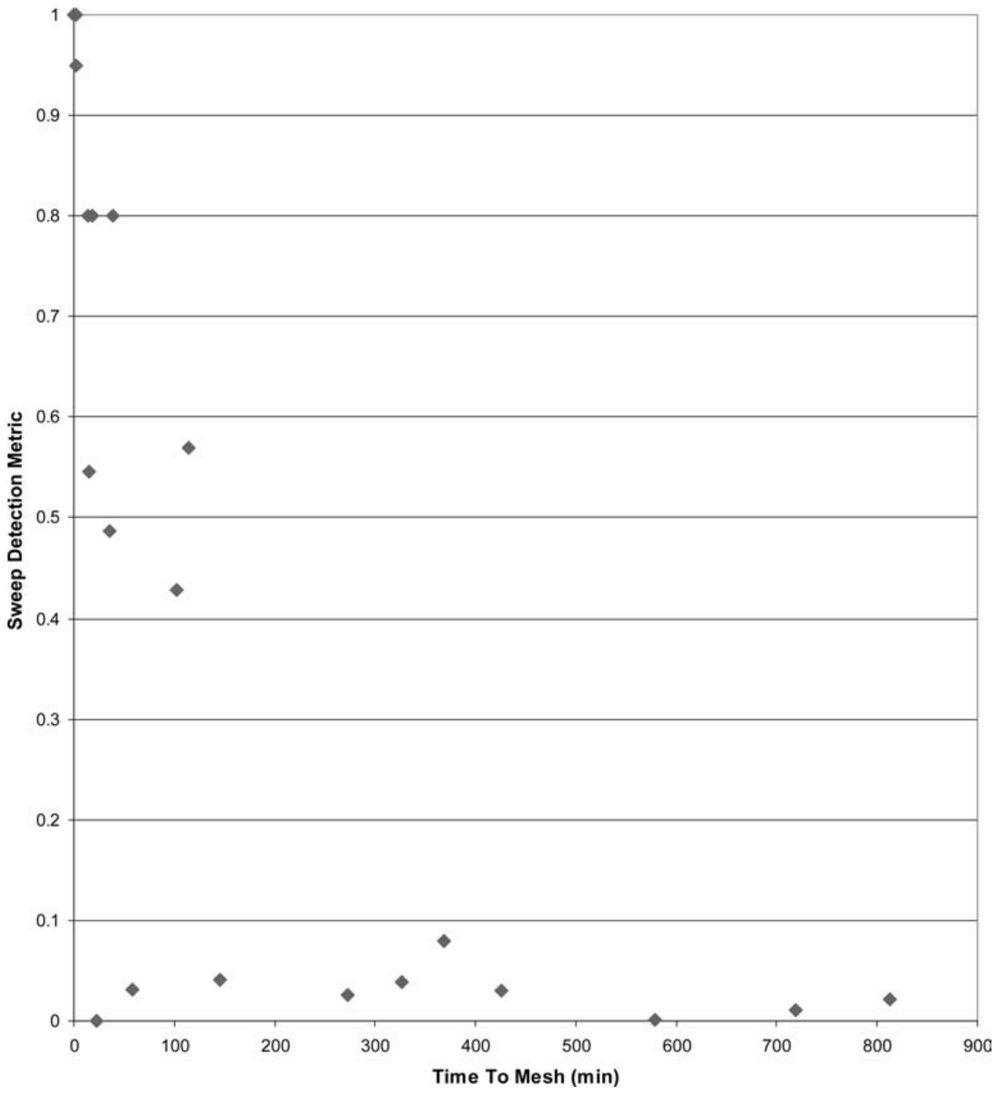


Fig. 11 Sweep detection metric versus time to mesh (min)

mesh between various CAD parts. Two kinds of metrics have been identified; they are called base metrics and negative metrics, respectively. Base metrics analyze the shape of the parts while negative metrics detect features that traditionally cause problems with mesh generation. Three base metrics have been identified: inverse topology count, sweep detection and Cartesian edges. The inverse topology count metric captures complexity that typically exists as the number of entities in the model increases. The sweep detection metric identifies the degree to which the model can be swept by inspection of both the topology and geometry. The Cartesian edges metric measures the level to which the model is blocky. Five negative metrics are proposed: number of small edges, faces and close face loops; number of small and large angles; number of bad CAD definitions; number of groups of blend faces; and the number of tangential face intersections. The number of small edges, faces and close face loops metric is determined by the element size provided by the user and identifies small regions that are difficult to mesh. The number of bad CAD definitions metric is obtained by identifying the bad geometric definitions that can impede mesh generation. The number of groups of blend faces and the number of tangential face intersections metrics identify the often harmful blend faces and tangential intersections through sets of proposed rules. The base and negative metrics are combined to form the proposed metric.

The proposed complexity metric is compared with the times to mesh using a set of 24 test cases consisting of real parts. The metric correlates well with the timing data. For some parts the metric does not correlate accurately due to both the limitations of the metric and the difficulty in predicting human computer interactions.

The proposed metric is intended to be used for two reasons. First, by analysts and managers to aid in predicting the time it will take to generate a mesh on certain CAD models. Based on previously meshed models, and the metric values computed for them, analysts and mangers can compute the complexity of new models allowing them to predict the time required for mesh generation. Finally, the proposed metric could also be used to help measure progress that is made in research in unstructured hexahedral mesh generation.

Acknowledgements The authors wish to acknowledge Sandia National Laboratories for providing funding and motivation for this project. Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000. Additionally, we wish to thank ANSYS Inc. for providing many of the CAD models that were included in the test suite.

References

1. Butlin G, Stops C (1996) CAD data repair. In: Proceedings of 5th international meshing roundtable, pp 7–12
2. Cheney D (1998) CAD model quality holds the key for analysis. In: Proceedings 7th International Meshing Roundtable, pp 539–546
3. Mezentsev A (1999) Methods and Algorithms of Automated CAD Repair for Incremental Surface Meshing. Proceedings 8th International Meshing Roundtable, pp 299–309
4. http://www.spatial.com, April 2003
5. http://www.eds.com/products/plm/parasolid/portfolio/bodyshop.shtml, April 2003
6. http://www.cadfix.com, April 2003
7. http://www.cadiq.com, April 2003
8. White D, Leland R, Saigal S, Owen S (2001) The meshing complexity of a solid: an introduction. In: Proceedings of 10th International meshing roundtable, pp 373–384
9. Steinbrenner J, Wyman N, Chawner J (2000) Fast surface meshing on imperfect cad models. In: Proceedings 9th International meshing roundtable. pp 33–41
10. Marcum D, Gaither A (1999) Unstructured surface grid generation using global mapping and physical space approximation. In: Proceedings 8th International meshing roundtable, pp 397–406
11. Sheffer A, Blacker T, Clements J, Bercovier M (1997) Virtual topology operators for meshing. In: Proceedings 6th International meshing roundtable, pp 49–66
12. Sheffer A, Blacker T, Bercovier M (1997) Clustering: automated detail suppression using virtual topology. Trends in unstructured mesh generation. ASME 220:57–64
13. Armstrong C, Bridgett S, Donaghy R, McCune R, McKeag R, Robinson D (1998) Techniques for interactive and automatic idealisation of CAD models. Num Grid Generation Comp Field Sim, pp 643–662
14. Blacker T, Sheffer A, Clements J, Bercovier M (1997) Using virtual topology to simplify the mesh generation process. Trends in unstructured mesh generation. ASME 200:45–50
15. Tautges T (2001) Automatic detail reduction for mesh generation applications. In: Proceedings 10th International meshing roundtable, pp 407–418
16. Mobley A, Carroll M, Canann S (1998) An object oriented approach to geometry defeaturing for finite element meshing. In: Proceedings 7th International meshing roundtable, pp 547–563
17. Armstrong C, Robinson D, McKeag R, Li T, Bridgett S, Donaghy R, McGleenan C (1995) Medials for meshing and more. In: Proceedings 4th International meshing roundtable, pp 277–288
18. Sheffer A, Etzion M, Rappoport A, Bercovier M (1998) Hexahedral mesh generation using the embedded voronoi graph. In: Proceedings 7th International meshing roundtable, pp 347–364
19. Lu Y, Gadh R, Tautges TJ (1999) Volume decomposition and feature recognition for hexahedral mesh generation. In: Proceedings 8th International meshing roundtable, pp 269–280
20. Schneiders R, Schindler R, Weiler F (1996) Octree-based generation of hexahedral element meshes. In: Proceedings 5th International roundtable pp 205–216
21. Tautges T, Blacker T, Mitchell S (1996) The whisker weaving algorithm: a connectivity-based method for constructing all-hexahedral finite element meshes. Int J Num Methods Eng 39:3327–3349
22. Folwell N, Mitchell S (1998) Reliable whisker weaving via curve contraction. In: Proceedings 7th International meshing roundtable, pp 365–378
23. Blacker T, Meyers R (1993) Seams and wedges in plastering: a 3D hexahedral mesh generation algorithm. Eng Comput 2:83–93
24. Mitchell S (1998) The all-hex geode-template for conforming a diced tetrahedral mesh to any diced hexahedral mesh. In: Proceedings 7th International meshing roundtable, pp 295–305
25. Muller- Hannemann M (1998) Hexahedral mesh generation by successive dual cycle elimination. In: Proceedings 7th International meshing roundtable, pp 365–378
26. Ymakawa S, Shimada K (2001) Hexhoop: modular templates for converting a hex-dominant mesh to an all-hex mesh. In: Proceedings 10th International meshing roundtable, pp 235–246
27. Cook W, Oaks W (1983) Mapping methods for generating three-dimensional meshing. Comput Mech Eng 1:67–72
28. White D, Mingwu L, Benzley S, Sjaardema G (1995) Automated hexahedral mesh generation by virtual decomposition. In: Proceedings 4th International meshing roundtable, pp 165–176
29. Blacker T (1996) The cooper tool. In: Proceedings 5th International meshing roundtable, pp 13–30
30. White D, Tautges T (2000) Automatic scheme selection for toolkit hex meshing. Int J Num Methods Eng 49:127–144
31. Mitchell S (1997) High fidelity interval assignment. In: Proceedings 6th International meshing roundtable, pp 33–44
32. Tautges T (2000) The common geometry module (CGM): a generic, extensible geometry interface. In: Proceedings 9th International meshing roundtable 337–348
33. Blacker T (1991) Paving: a new approach to automated quadrilateral mesh generation. Int J Num Methods Eng 32:811–847
34. Owen S, Staten M, Canann S, Saigal S (1999) Q-Morph: an indirect approach to advancing front quad meshing. Int J Num Methods Eng 44:1317–1340
35. Mitchell S (1997) Choosing corners of rectangles for mapped meshing. In: 13th Annual symposium on computational geometry, ACM Press, pp 87–93
36. Shepherd J (2003) CUBIT mesh generation toolsuite. http://cubit.sandia.gov.